# 패키지의 활용 (Using Packages)

델파이에서는 델파이의 버전 3 에서부터 패키지(package)라는 새로운 개념을 도입함으로써 보다 효율적이고, 분산된 구조를 가질 수 있게 되었다. 패키지란 간단히 말하면 컴포넌트를 모아서 이들을 적절하게 분산할 수도 있고, 어플리케이션을 작게 만들 수도 있는 DLL wrapper 이다.
엄밀히 말하면 전혀 새로운 개념이라고는 할 수 없는 것이 패키지이다. 과거에도 델파이에서 DLL 을 이용해서 어플리케이션을 여러 개로 분산시킬 수 있었다. 그렇지만 패키지는 이 보다 훨씬 더 편리하고, 진보된 개념이다.
이번 장에서는 패키지 개념에 대해서 알아보고, 이를 실제 어플리케이션에서 어떻게 이용할 수 있는지 알아보도록 한다.

## 패키지(Package)란 ?

패키지란 일종의 DLL 로, DLL 에 보다 많은 장점을 부여한 것으로 생각하면 된다. 패키지 파일의 확장자는 BPL 인데, BPL 파일을 사용할 때의 장점에는 다음과 같은 것들이 있다. 참고로 델파이 3 버전에서의 BPL 에 해당되는 확장자는 DPL 이다.

- 실행 파일의 크기를 줄여 준다.
- 어플리케이션을 배포하고 업데이트하기 쉽다.
- 여러 개의 어플리케이션에서 리소스를 공통적으로 사용할 수 있다.

DLL 에 비해 확실한 장점으로 꼽을 수 있는 것은 어플리케이션의 조각을 쉽게 합쳐서 패키지 모듈로 재구성할 수 있다는 것이다. 또한, 기본적으로 바뀌지 않는 부분은 내버려 두고 바뀐 부분의 모듈만을 배표할 수도 있다. DLL 과는 달리 패키지를 사용할 때에는 함수를 미리 선언하지 않아도 되고, 명시적으로 로드할 필요도 없다. 단지 필요한 것은 uses 절에 모듈의 이름을 적어주는 것으로 충분하다.
패키지와 관련된 파일 확장자에는 다음과 같은 것들이 있다.

| 확장자 | 설 명 |
|---|---|
| DPK | 패키지에 담겨 있는 유닛들에 대한 소스 파일 |
| DCP | 패키지 헤더와 컴파일러가 요구하는 심볼 정보를 포함한 패키지의 모든 DCU 파일 들이 하나로 합쳐진 바이너리 이미지 파일로 패키지당 하나씩 생성된다. |
| DCU | 패키지에 포함된 유닛 파일의 바이너리 이미지 파일 |
| BPL | 런타임 패키지로 일종의 윈도우 DLL 파일이다. |

## 패키지의 필요성

기본적으로 델파이의 모든 기본적인 컴포넌트는 패키지로 구성되어 있다. 델파이의 표준 컴포넌트 라이브러리에 패키지를 적용함으로써 얻을 수 있는 효과도 많지만, 실제로 더욱 도움이 된 것은 써드 파티에서 개발한 컴포넌트 들을 패키지를 이용해서 대단히 쉽게 설치, 이용할 수 있게 된 것이다.
개발자가 어플리케이션을 사용할 때 상당히 많은 수의 써드 파티 라이브러리를 사용한다고 가정하자. 이렇게 되면 실행 파일의 크기가 1MB 는 훌쩍 뛰어 넘기가 일수이다. 이렇게 커다란 실행 파일의 대부분을 차지하는 것은 코드에 포함된 정적 라이브러리의 코드 들이다. 그러나 이들 중 빈번히 사용되고, 변경하는 부분은 그다지 많지 않은 경우가 대부분이다. 만약에 이들을 분리된 패키지 파일로 분리해낼 수 있으면 실행 파일의 크기를 많이 줄일 수 있음은 쉽게 미루어 짐작할 수 있을 것이다.
단적인 예를 들어, 폼에 TTable, TDataSource, TDBGrid, TDBNavigator 컴포넌트를 하나씩 얹고, 어플리케이션을 컴파일하면 실행파일의 크기는 400KB 를 넘게 된다. 그렇지만, 이를 패키지를 이용해서 분리해 주면 약 13KB 정도의 아주 작은 실행 파일로 만들어낼 수 있게 된다.
패키지를 활용할 수 있는 또 다른 좋은 상황은 동일한 모듈을 포함한 여러 개의 실행 파일을 만들 때이다. 예를 들어, 동일한 컴포넌트 들을 사용해서 개발한 여러 개의 어플리케이션이 있다면 이들은 동일한 패키지를 공유할 수 있다. 이럴 때에 여러 개의 어플리케이션을 각각 업그레이드 하기 보다, 동일한 부분의 기능을 향상 시켜서 배포할 수도 있게 된다.
DLL 과 패키지의 또 다른 점은 패키지 파일은 프로젝트에 정적으로 링크된다는 점이다. 런타임에는 사용된 모든 DPL 파일이 일단 메모리에 적재된다. 그렇기 때문에, 패키지는 DLL 과 같은 메모리 절약 효과는 없다.
패키지를 사용하는 것이 꼭 장점만 있는 것은 아니다. 기본적으로 조각을 내어 업그레이드를 하는 것이므로 버젼 컨트롤(version control)에 보다 많은 신경을 써야 한다.

## 패키지의 종류

패키지에는 다음과 같이 세가지가 있다.

- 런타임 패키지 (Runtime Packages)

가장 흔히 사용되는 형태로 어플리케이션을 배포할 때 같이 배포되는 것이다. 여기에는 컴포넌트나 함수 라이브러리가 포함된다.

- 디자인 타임 패키지 (Design-time Packages)

델파이의 IDE 에 컴포넌트를 인스톨하고, 이들 컴포넌트에 대한 프로퍼티 에디터(property editor)를 포함할 때 사용하는 패키지로 IDE 자체가 워낙 커다란 어플리케이션이기 때문에 이들을 유용하게 이용할 수 있다. 그러므로, 실제로 잘 쓰이지 않는 컴포넌트를 패키지로 분리해 두었다가, 필요할 때 사용하는 것도 좋은 아이디어가 될 수 있다. 기본적으로 패키지는 런타임 패키지 이면서 동시에 디자인 타임 패키지일 수도 있다.

- 커스텀 패키지 (Custom Packages)

커스텀 패키지는 개발자가 직접 패키지로 개발한 것을 말한다. 여기에는 각종 컴포넌트와 클래스, 함수 라이브러리가 포함될 수 있으며, 어플리케이션에 쉽게 포함될 수 있다.

## 패키지의 설치

컴포넌트를 설치하기 위해 델파이의 메뉴를 살펴 보면, 과거 델파이 2.0 까지 존재하던 Install Component 메뉴이외에 Install Packages 메뉴가 추가되어 있는 것을 확인할 수 있을 것이다. 모든 볼랜드사의 컴포넌트와 델파이 4 에 번들로 들어온 써드 파티 컴포넌트가 모두 패키지로 구성되어 있다. 과거에는 컴포넌트를 설치하기 위해 유닛 파일(.PAS 또는 .DCU)을 이용했지만, 패키지를 설치할 때에는 BPL, DCP 파일을 사용하게 된다. 하나의 컴포넌트를 설치해도 반드시 패키지 파일을 지정해 주어야 한다.

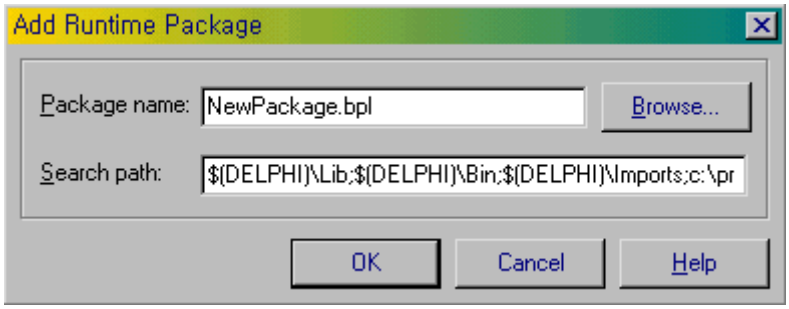
Component 메뉴에서 Install Packages 를 선택하면 IDE 에 설치될 디자인 타임 패키지를 선택할 수 있는 대화 상자가 나타난다. 여기에서 델파이 4 의 현재 컴포넌트 팔레트에 추가할 패키지를 선택할 수 있다. ‘Add’ 버튼을 클릭하면 추가할 패키지 파일을 선택할 수 있다.

패키지 파일과 함께 .DPC 라는 확장자를 가지는 파일도 발견할 수 있는데, 이 파일의 확장자는 ‘Delphi Package Collection’의 약자로 어플리케이션에 의해 하나 이상의 패키지가 사용될 때 이들을 모아 놓은 파일을 의미한다.

## 어플리케이션에 패키지 사용하기

어플리케이션에 패키지를 사용하려면, 패키지를 사용할 어플리케이션 프로젝트의 Project|Options 메뉴를 선택하고, Packages 탭에서 Build with Runtime Packages 체크박스를 선택하고, 하나 이상의 패키지의 이름을 적어 넣으면 된다. 이때 디자인-타임 패키지와 연결된 런타임 패키지는 이미 에디트 박스에 나열되어 있을 것이다.

이미 존재하는 리스트에 패키지를 추가하려면, Add 버튼을 클릭하고 새로운 패키지의 이름을 다음과 같은 Add Runtime Package 대화 상자에서 입력하면 된다. 이때 Browse 를 선택하여 패키지 파일의 위치를 지정할 수 있다.




Runtime Packages 에디트 박스에 나열된 런타임 패키지 들은 자동적으로 컴파일 시에 어플리케이션과 링크되는 패키지 들이다. 이 내용이 비어있으면, 어플리케이션이 패키지 없이 컴파일 된다.
런타임 패키지는 현재 프로젝트에 대해서만 선택된다. 이때, 선택한 패키지를 앞으로의 프로젝트에 자동으로 디폴트로 사용하게 하려면 Defaults 체크 박스를 선택하면 된다.

- 어떤 런타임 패키지를 사용할 것인가 ?

델파이에는 여러 가지 기본적인 언어와 컴포넌트를 지원하기 위한 런타임 패키지가 지원된다. 가장 흔히 사용되는 것은 VCL40 으로 가장 기본적인 컴포넌트와 시스템 함수, 윈도우 인터페이스를 제공한다. 다음에 델파이에서 제공되는 런타임 패키지에서 지원되는 유닛에 대해서 나열해 보았다.

| 패키지 | 포함된 유닛 |
|---|---|
| VCL40.BPL | Ax, Buttons, Classes, Clipbrd, Comctrls, Commctrl, Commdlg, Comobj, Comstrs, Consts, Controls, Ddeml, Dialogs, Dlgs, Dsgnintf, Dsgnwnds, Editintf, Exptintf, Extctrls, Extdlgs, Fileintf, Forms, Graphics, Grids, Imm, IniFiles, Isapi, Isapi2, Istreams, Libhelp, Libintf, Lzexpand, Mapi, Mask, Math, Menu, Messages, Mmsystem, Nsapi, Ole2I, Oleconst, Olectnrs, Olectrls, Oledlg, Penwin, Printers, Proxies, Registry, Regstr, Richedit, Shellapi, Shlobj, Stdctrls, Stdvcl, Sysutils, Tlhelp32, Toolintf, Toolwin, Typinfo, Vclcom, Virtintf, Windows, Wininet, Winsock, Winspool, Winsvc |
| VCLX40.BPL | Checklst, Colorgrd, Ddeman, Filectrl, Mplayer, Outline, Tabnotbk, Tabs |
| VCLDB40.BPL | Bde, Bdeconst, Bdeprov, Db, Dbcgrids, Dbclient, Dbcommon, Dbconsts, Dbctrls, Dbgrids, Dbinpreq, Dblogdlg, Dbpwdlg, Dbtables, Dsintf, Provider, SMintf |
| VCLDBX40.BPL | Dblookup, Report |
| DSS40.BPL | Mxarrays, Mxbutton, Mxcommon, Mxconsts, Mxdb, Mxdcube, Mxdssqry, Mxgraph, Mxgrid, Mxpivsrc, Mxqedcom, Mxqparse, Mxqryedt, Mxstore, Mxtables, Mxqvb |
| QRPT40.BPL | Qr2const, Qrabout, Qralias, Qrctrls, Qrdatasu, Qrexpbld, Qrextra, Qrprev, |

| | |
|---|---|
| | Qrprgres, Qrprntr, Qrqred32, Quickrpt |
| TEE40.BPL | Arrowcha, Bubblech, Chart, Ganttch, Series, Teeconst, Teefunci, Teengine, Teeprocs, Teeshape |
| TEEDB40.BPL | Dbchart, Qrtee |
| TEEUI40.BPL | Areaedit, Arrowedi, Axisincr, Axmaxmin, Baredit, Brushdlg, Bubbledi, Custedit, Dbeditch, Editchar, Flineedi, Ganttedi, leditcha, Pendlg, Pieedit, Shapeedi, Teeabout, Teegally, Teelisb, Teeprevi, Teexport |
| VCLSMP40.BPL | Sampreg, Smpconst |

- 패키지의 동적 적재

패키지를 동적으로 어플리케이션에 적재하려면 LoadPackage 함수를 사용하면 된다. 예를 들어, 다음의 코드는 파일 열기 대화 상자에서 선택한 파일을 어플리케이션에 적재한다.

```
with OpenDialog do
  if Execute then
    with PackageList.Items do
      AddObject(FileName, Pointer(LoadPackage(FileName)));
```

반대로, 동적으로 패키지를 메모리에서 내리고 싶을 때에는 UnloadPackage 프로시저를 사용하면 된다. 그렇지만, 패키지를 메모리에서 해제하려면 패키지와 연관된 객체와 클래스를 먼저 모두 파괴해야 하는 것을 명심하기 바란다.

- 디자인-타임 패키지

디자인-타임 패키지는 IDE 의 컴포넌트 팔레트에 컴포넌트를 추가하고, 컴포넌트에 대한 프로퍼티 에디터를 생성할 때 사용된다.
델파이 4 에는 다음과 같은 디자인-타임 컴포넌트 패키지가 IDE 에 설치되어 있다.

| 패키지 | 컴포넌트 팔레트 | 패키지 | 컴포넌트 팔레트 |
|---|---|---|---|
| DCLSTD40.BPL | Standard, Additional, System, Win32, Dialogs | DCLTEE40.BPL | Additional (TChart component) |
| DCLDB40.BPL | Data Access, Data Controls | DCLMID40.BPL | Data Access (MIDAS) |
| DCL31W40.BPL | Win 3.1 | NMFAST.BPL | Internet |
| DCLSMP40.BPL | Samples | DCLOCX40.BPL | ActiveX |
| DCLQRT40.BPL | QReport | DCLDSS40.BPL | Decision Cube |

| | | | |
|---|---|---|---|
| IBSMP40.BPL | Samples<br>(IBEventAlerter component) | DCLINT40.BPL | International Tools<br>(Resource DLL wizard) |

이들 디자인-타임 패키지는 Requires 절에 참조되어 있는 런타임 패키지를 호출하여 동작한다. 예를 들어, DCLSTD40 은 VCL40 런타임 패키지를 호출한다.

## 패키지 소스 파일

분리된 소스 파일에서 선언된 패키지는 DPK 파일로 저장된다. 패키지 소스 파일은 다음과 같은 형태를 가지고 있다.

```
package packageName;
  requiresClause;
  containsClause;
end.
```

여기서 requiresClause, containsClause 는 옵션이다. 예를 들어, 다음 코드는 VCLDB40 패키지를 선언하는 것이다.

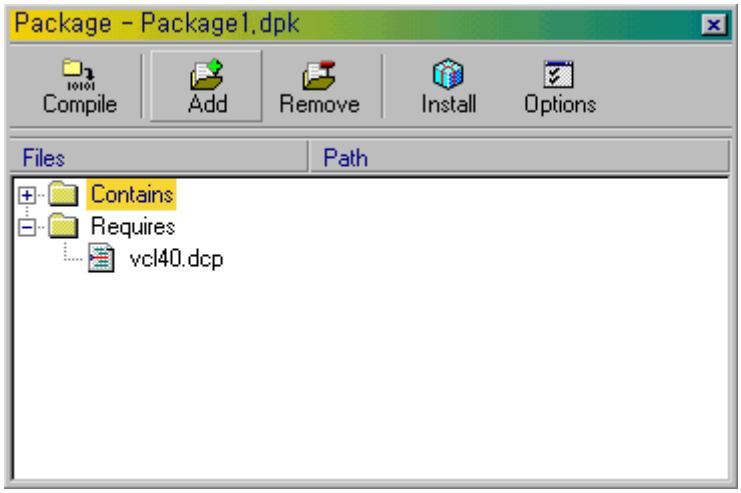
```
package VCLDB40;
  requires VCL40;
  contains Db, Dbcgrids, Dbctrls, Dbgrids, ... ;
end.
```

이와 같이 requires 절에는 패키지에서 사용하는 다른 외부 패키지를 선언한다. 만약 다른 패키지를 참조하지 않으면, require 절이 필요하지 않다. contains 절에는 패키지에 컴파일 될 유닛이 지정된다. 경우에 따라서는 다음과 같이 디렉토리 패스를 지정할 수도 있다.

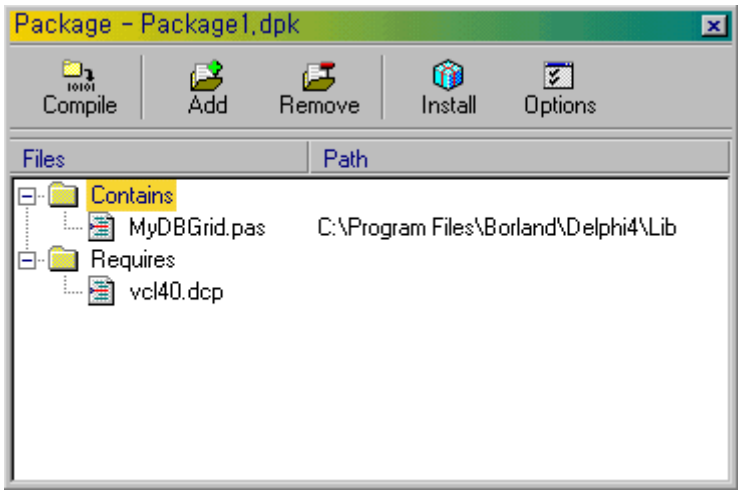
```
contains MyUnit in 'C:₩MyProject₩MyUnit.pas';
```

## 패키지 만들기

패키지를 만들려면, File|New 메뉴를 선택하고 package 를 선택한다. 그러면, 다음 그림과 같이 패키지 에디터가 뜬다.




File|Save As 명령을 선택하여, 적당한 패키지 이름을 정해서 저장한다. 이때 파일은 .DPK 확장자를 가지게 된다. 패키지에 유닛을 추가하려면, Add 버튼을 누르고 추가할 유닛 파일이나 액티브 X 컴포넌트 등을 지정하면 된다. 다음은 필자가 작성한 컴포넌트 유닛을 하나 추가한 화면이다.

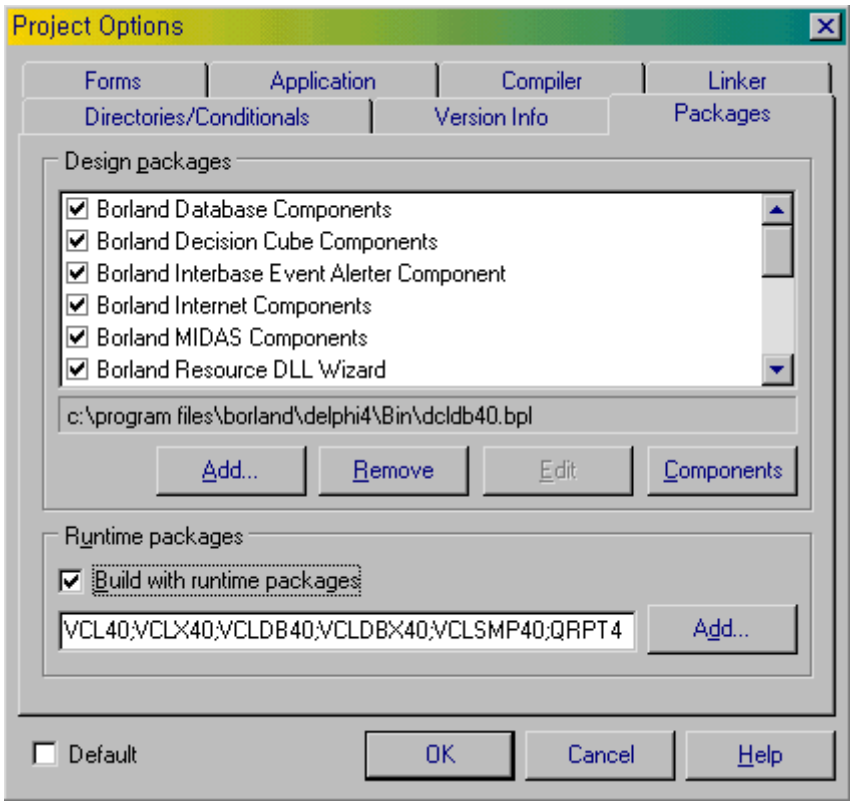



일단 이렇게 패키지에 파일과 컴포넌트를 모두 추가 했으면 컴파일을 한다. 컴파일이 정상적으로 완료되면 BPL 파일이 생성되며 Install 버튼을 선택하면 델파이의 IDE 에서 사용할 수 있게 된다. 패키지의 Options 메뉴에서 몇 가지 옵션을 줄 수 있는데, 예를 들어 컴파일할 때마다 패키지를 rebuild 하게 하거나, build 를 선택했을 때에만 패키지가 컴파일 되게 하는 등의 선택을 할 수 있다.

## 패키지의 배포

기본적으로 델파이 4 는 모든 어플리케이션 제작에 사용된 모든 컴포넌트와 유닛을 하나의

실행파일에 통합하게 된다. 이런 구조에서 패키지를 이용하는 형태로 전환하려면 Project|Options 메뉴 아이템을 선택해서 패키지를 이용하도록 설정해 주어야 한다.
Project|Options 메뉴 아이템을 선택하면 지금 현재 사용 중인 프로젝트에 대한 여러가지 옵션을 선택할 수 있다. 여기에서 Packages 탭을 클릭하면 패키지에 대한 여러가지 설정을 할 수 있게 된다. 이를 선택한 화면은 다음 그림과 같은데, Component|Install Packages 를 선택했을 때와 거의 비슷한 형태 임을 알 수 있을 것이다.

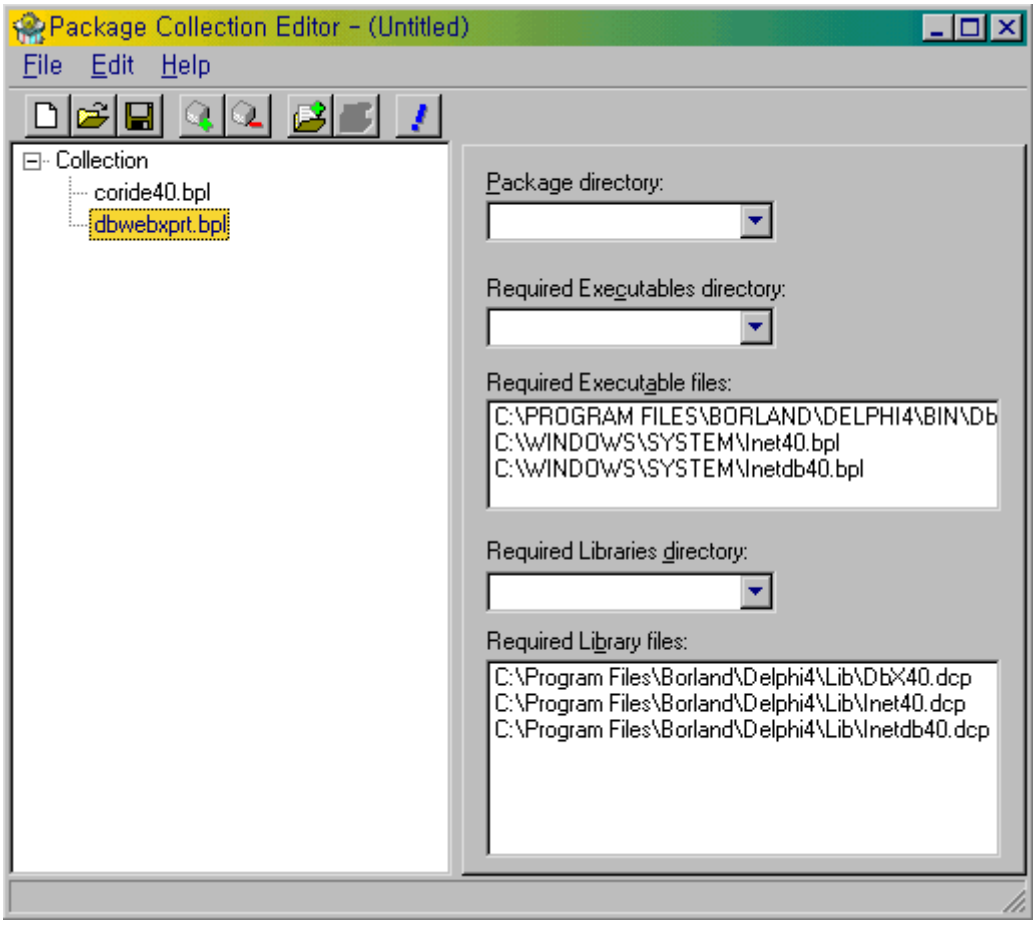



여기에서 보이는 Design Packages 목록에는 컴포넌트 팔레트에 나타나는 디자인 타임 패키지가 선택된다. 그렇지만, 패키지를 배포할 때에는 아래에 있는 런타임 패키지를 선택하는 것이 중요한데, 'Build with runtime packages' 체크 박스를 선택하고, 같이 배포할 런타임 패키지를 여기에 Add 하면 하나의 실행 파일로 만들어질 부분 들이 패키지로 분할된다.

## 패키지 컬렉션 (Package Collections)

패키지 컬렉션은 패키지 컬렉션 에디터 유틸리티로 제작한다. 패키지 컬렉션이란 여러 개의 패키지를 하나로 묶어서 편리하게 사용할 수 있게 한 것이다. 필요한 모든 유닛과 패키지를 이를 이용해서 하나로 묶어서 여러 가지의 다른 프로젝트를 하나의 단순한 작업으로 쉽게 작업할 수 있는 장점이 있다.

그러면, 실제로 패키지 컬렉션을 만드는 작업에 대해서 알아보자.
델파이의 Tools 메뉴에서 Package Collection Editor 를 선택하면 다음 그림과 같은 유틸리티 프로그램이 실행된다. 여기에서는 Edit|Add Package 메뉴를 선택해서 두개의 패키지를 추가하였다. 패키지를 모두 추가하고, 패키지 파일의 위치와 필요한 라이브러리의 위치를 지정해 주고 나면 이를 컴파일 한다. 이렇게 하면 DPC 파일이 생성되는데, 이 파일을 이용해서 여러 개의 패키지를 한꺼번에 사용할 수 있게 된다.




## 이 장을 마치며 ...

패키지란 것이 그다지 복잡하지는 않지만, 개발자에게 주는 이득은 상당한 것이다. 지금까지 살펴 보았듯이 패키지를 이용하는 방법은 어플리케이션을 개발할 때 사용하는 컴포넌트를 추가, 삭제, 그리고 지정하는 단순한 작업이다.
디자인 타임 패키지를 잘 이용하면 델파이를 이용해서 개발을 할 때 많은 유연성을 발휘할 수 있다. 런타임 패키지는 어플리케이션의 중복되는 코드 부분을 줄여줄 수 있도록 활용할 수 있으며, 자칫 비대해지기 쉬운 실행 파일을 작게 유지시킬 수도 있다. 뿐만 아니라 적절한 버젼 컨트롤이 가능하다면 업그레이드와 유지 보수에도 매우 편리하게 활용할 수 있다.